Научно-исследовательский практикум

ПОЛИМОРФИЗМ. МНОЖЕСТВЕННОЕ НАСЛЕДОВАНИЕ.

Базовый класс

```
class Base
{
public:
        Base();
        ~Base();
        void NonVirtualMethod();
        virtual void VirtualMethodA();
        virtual void VirtualMethodB();
};
Base::Base()
{
        cout << "Base default c-tor called" << endl;
}

Base::~Base()
{
        cout << "Base d-tor called" << endl;
}

void Base::NonVirtualMethod()
{
        cout << "Non-virtual method of Base class called" << endl;
}

void Base::VirtualMethodA()
{
        cout << "Virtual method A of Base class called" << endl;
}

void Base::VirtualMethodB()
{
        cout << "Virtual method B of Base class called" << endl;
}
```

Первый наследник

Здесь у нас поменялись конструктор и деструктор – теперь мы в них выделяем память

```
class HeirA : public Base
{
public:
        HeirA();
        ~HeirA();
        void NonVirtualMethod();
        virtual void VirtualMethodA();
        int* data;
};

void HeirA::NonVirtualMethod()
{
        cout << "Non-virtual method of Heir A class called"
<< endl;
}
void HeirA::VirtualMethodA()
{
        cout << "Virtual method A of Heir A class called" <<
endl;
}
HeirA::HeirA()
{
        cout << "Creating HeirA object..." << endl;
        data = new int [8];
        cout << "HeirA created and dynamic memory allocated"
<< endl;
}
HeirA::~HeirA()
{
        cout << "Heir d-tor called" << endl;
        delete data;
        cout << "Dynamic memory dealocated" << endl;
}
```

main()

Теперь объекты в динамической памяти

```
int main(int argc, char *argv[])
{
        Base* ptr = nullptr;

        ptr = new Base;
        ptr->NonVirtualMethod();
        ptr->VirtualMethodA();
        ptr->VirtualMethodB();
        delete ptr;

        ptr = new HeirA;
        ptr->NonVirtualMethod();
        ptr->VirtualMethodA();
        ptr->VirtualMethodB();

        return 0;
}
```

Что получилось

Base

HeirA

```
Starting /Users/amakashov/projects/build-heir_example2-
Base default c-tor called
Non-virtual method of Base class called
Virtual method A of Base class called
Virtual method B of Base class called
Base d-tor called
Base default c-tor called
Creating HeirA object...
HeirA created and dynamic memory allocated
Non-virtual method of Base class called
Virtual method A of Heir A class called
Virtual method B of Base class called
Base d-tor called
/Users/amakashov/projects/build-heir_example2-Desktop-D
```

Мы не вызвали деструктор класса-наследника

В результате у нас «утекла» память (а могло и что-нибудь похуже случиться)

Сделаем виртуальный деструктор

Вообще, не обязательно вручную прописывать virtual для деструкторов наследников – если методы виртуальный в базовом классе, то он везде виртуальный

```
class Base
{
public:
	Base();
	virtual ~Base();
	virtual void VirtualMethodA();
	virtual void VirtualMethodB();

	void NonVirtualMethod();
};

class HeirA : public Base
{
public:
	HeirA();
	virtual ~HeirA();

	void NonVirtualMethod();
	virtual void VirtualMethodA();
	int* data;
};
```

Что получилось

Base

HeirA

```
Starting /Users/amakashov/projects/build-heir_example2
Base default c-tor called
Non-virtual method of Base class called
Virtual method A of Base class called
Virtual method B of Base class called
Base d-tor called
Base default c-tor called
Creating HeirA object...
HeirA created and dynamic memory allocated
Non-virtual method of Base class called
Virtual method A of Heir A class called
Virtual method B of Base class called
Heir d-tor called
Dynamic memory dealocated
Base d-tor called
/Users/amakashov/projects/build-heir_example2-Desktop-
```

Чистые виртуальные функции

Создадим ***абстрактный*** класс:

```
class Base
{
public:
	Base() {}
	virtual ~Base() {}

	virtual void MakeData(int first, int second) = 0;
	virtual void PrintResult() = 0;
};
```

Первый наследник

Здесь у нас простейшая арифметическая операция

```
class Calculate : public Base
{
public:
        void MakeData(int first, int second);
        void PrintResult();
protected:
        int m_result;
};

// В *.cpp-файле
using std::cout;
using std::endl;

void Calculate::MakeData(int first, int second)
{
        m_result = first * second;
}

void Calculate::PrintResult()
{
cout << "CALCULATE : Result is " << m_result << endl;
}
```

Второй наследник

А здесь мы будем делать строку

```
class Concatenate : public Base
{
public:
        void MakeData(int first, int second);
        void PrintResult();
protected:
        std::string m_string;
};
void Concatenate::MakeData(int first, int second)
{
        m_string = "First is " + std::to_string(first)
        + " and second " + std::to_string(second);
}

void Concatenate::PrintResult()
{
        cout << "CONCATENATE: "<< m_string << endl;
}
```

Третий наследник

Тут мы будем рисовать плюсики

```
class Draw : public Base
{
public:
        void MakeData(int first, int second);
        void PrintResult();
protected:
        int m_height=0,
            m_width=0;
};

void Draw::MakeData(int first, int second)
{
        m_height = first;
        m_width = second;
}

void Draw::PrintResult()
{
        cout << "DRAW:"<< endl;
        for (int i=0; i< m_height; i++)
        {
                for (int j=0; j<m_width; j++)
                        cout << "+";
                cout << endl;
        }
        cout << endl;
}
```

main.cpp

Функция DoSomething() принимает на вход ссылку

```
void DoSomething (Base& classRef)
{
        classRef.MakeData(4, 18);
        classRef.PrintResult();
}

int main(int argc, char *argv[])
{
        // Base baseClass; // ошибка – нельзя создать экземпляр
                           // абстрактного класса
        Calculate calcClass;
        Concatenate concClass;
        Draw drawClass;

        DoSomething(calcClass);
        DoSomething(concClass);
        DoSomething(drawClass);

        return 0;
}
```

Результат выполнения

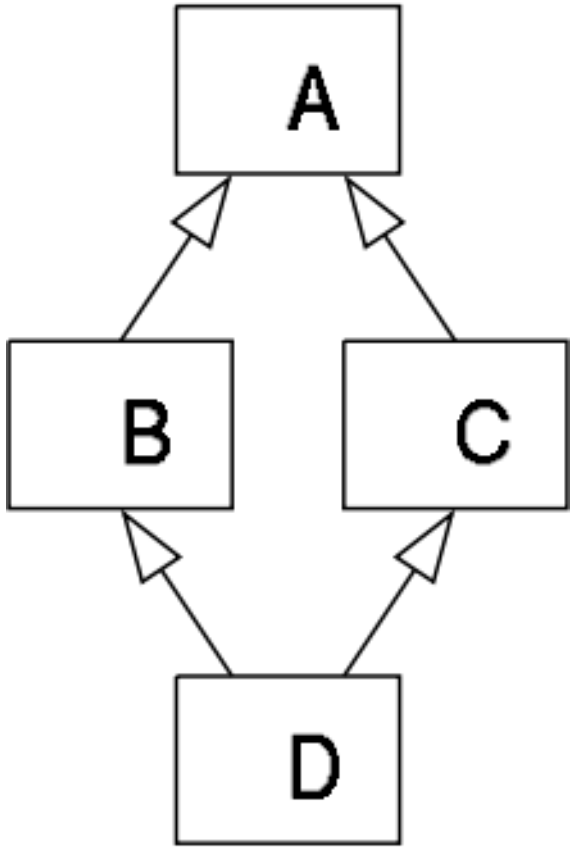
```
Starting /Users/amakashov/projects/build-heir_pure
CALCULATE : Result is 72
CONCATENATE: First is 4 and second 18
DRAW:
++++++++++++++++++
++++++++++++++++++
++++++++++++++++++
++++++++++++++++++

/Users/amakashov/projects/build-heir_pure_virtual-
```

Наследование: ромб (aka diamond)

Мы уже видели примеры с множественным наследованием

И даже пример с множественным наследованием мы видели (на 8 занятии)

Как оно выглядит с виртуальными методами?



Базовый класс

```
class Base
{
public:
        Base();
        ~Base();
        void NonVirtualMethod();
        virtual void VirtualMethodA();
        virtual void VirtualMethodB();
};
Base::Base()
{
        cout << "Base default c-tor called" << endl;
}

Base::~Base()
{
        cout << "Base d-tor called" << endl;
}

void Base::NonVirtualMethod()
{
        cout << "Non-virtual method of Base class called" << endl;
}

void Base::VirtualMethodA()
{
        cout << "Virtual method A of Base class called" << endl;
}

void Base::VirtualMethodB()
{
        cout << "Virtual method B of Base class called" << endl;
}
```

Первый наследник

```
class HeirA : public Base
{
public:
	void NonVirtualMethod();
	virtual void VirtualMethodA();
};

void HeirA::NonVirtualMethod()
{
	cout << "Non-virtual method of Heir A class called" << endl;
}

void HeirA::VirtualMethodA()
{
	cout << "Virtual method A of Heir A class called" << endl;
}
```

Второй наследник

```
class HeirB : public Base
{
public:
	void NonVirtualMethod();
	virtual void VirtualMethodB();
};

void HeirB::NonVirtualMethod()
{
	cout << "Non-virtual method of Heir B class called" << endl;
}

void HeirB::VirtualMethodB()
{
	cout << "Virtual method B of Heir B class called" << endl;
}
```

main.cpp

```cpp
int main(int argc, char *argv[])
{
        Base* ptr = nullptr;

        ptr = new Base;
        ptr->VirtualMethodA();
        ptr->VirtualMethodB();
        delete ptr;

        ptr = new HeirA;
        ptr->VirtualMethodA();
        ptr->VirtualMethodB();
        delete ptr;

        ptr = new HeirB;
        ptr->VirtualMethodA();
        ptr->VirtualMethodB();
        delete ptr;

        return 0;
}
```

Результат

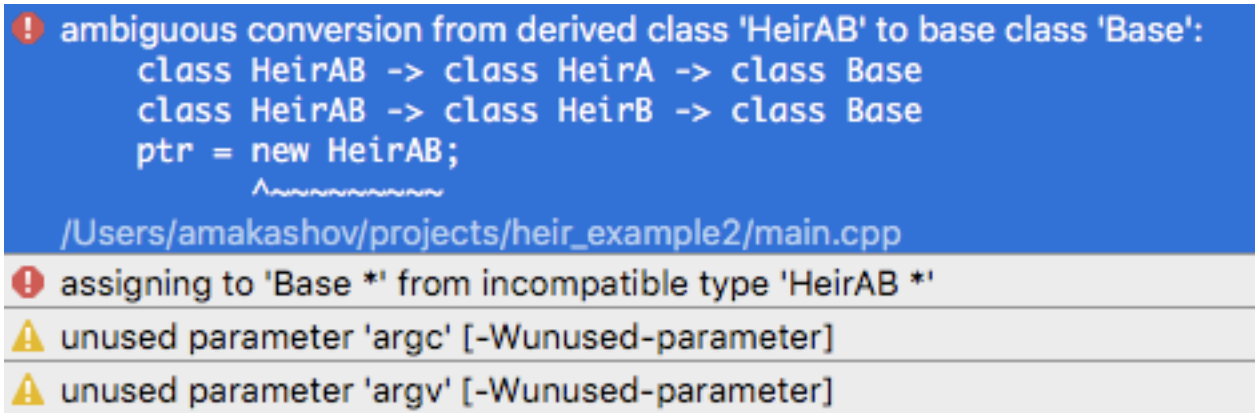
```
Starting /Users/amakashov/projects/build-heir_example2-
Virtual method A of Base class called
Virtual method B of Base class called
Virtual method A of Heir A class called
Virtual method B of Base class called
Virtual method A of Base class called
Virtual method B of Heir B class called
/Users/amakashov/projects/build-heir_example2-Desktop-
```

Ещё один наследник

```
class HeirAB : public HeirA, public HeirB
{
      virtual void VirtualMethodA();
      virtual void VirtualMethodB();
      void NonVirtualMethod();
};
```

Попробуем создать указатель такой объект...

```
ptr = new HeirAB;
ptr->VirtualMethodA();
ptr->VirtualMethodB();
delete ptr;
```



Самостоятельно

На прошлом занятии вы реализовали класс комплексных чисел.

Теперь давайте реализуем класс трёхмерных комплексных векторов:

1. Конструктор, принимающий три комплексных числа
2. Оператор [] для обращения к элементу
3. Операторы сложения, вычитания, скалярного и векторного умножения
4. Определите, нужны ли вам конструктор-копировщик, деструктор и оператор присваивания?